TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

TOPI HAAVISTO

OHJELMISTOTUOTANTOPROSESSIEN VAIKUTUS PROJEKTIIN

Kandidaatintyö

Tarkastaja: Pia Niemelä

TIIVISTELMÄ

Topi Haavisto: Ohjelmistotuotantoprosessien vaikutus projektiin
English title: Software development methodologies and their effect on a project
Tampereen teknillinen yliopisto
Kandidaatintyö, 17 sivua
Joulukuu 2017
Tietotekniikan kandidaatin tutkinto-ohjelma
Pääaine: Ohjelmistotuotanto
Tarkastaja: Pia Niemelä

Avainsanat: ohjelmistotuotantoprosessi, ohjelmistotuotannon menetelmät, elämänkaarimalli, projekti

Ohjelmistotuotantoprosesseja on nykyään monia erilaisia ja oikean prosessin valitseminen oikeaan projektiin voi olla vaikeaa. Tässä työssä tutkittiin ohjelmistotuotantoprosessien valinnan vaikutusta projektiin ja pyrittiin selvittämään, minkälaisiin projekteihin mikäkin käsitellyistä prosesseista eli vesiputousmallista, spiraalimallista, iteratiivisesta ja inkrementaalisesta mallista sekä ketteristä menetelmistä sopisi parhaiten. Työ toteutettiin tarkastelemalla tutkimusta, jossa valitut prosessit tai niitä vastaavat prosessit olivat käytössä sekä suorittamalla teoriapohjaista käsittelyä prosessien vahvuuksista, heikkouksista ja soveltuvuuksista tietynlaisiin projekteihin. Tutkimuksen tuloksina havaittiin, että käyttämällä ketterää menetelmää saadaan ohjelmointiprojektissa dokumentaation määrä minimoitua ja ohjelmakoodin määrä maksimoitua verrattuna muihin prosesseihin, joiden väliset keskinäiset erot osoittautuivat hillitymmiksi. Lisäksi tehtiin havaintoja prosessien soveltuvuudesta erilaisiin projekteihin. Koska oikean prosessin valitseminen ja siitä johtuvat vaikutukset ovat hyvin pitkälti projektikohtaisia, ovat tämän työn tulokset lähinnä suuntaa antavia neuvoja oikean prosessin valitsemiseen ja projektiin kohdistuvien vaikutusten arvioimiseen.

SISÄLLYSLUETTELO

1. JOHDANTO

Ohjelmistonkehitys on kehittynyt paljon viime vuosikymmenien aikana ja nykyään onkin olemassa jo useita erilaisia ohjelmistotuotannon prosesseja tai menetelmiä erilaisia ohjelmistoprojekteja varten. Ohjelmistokehityksen organisaatioilla on mahdollisuus valita sopivin menetelmä perinteisten menetelmien ja uudempien ketterää kehitystä edustavien menetelmien väliltä. Vaikka ketterän kehityksen menetelmät ovat nykyään aiempaa suuremmassa suosiossa, ovat perinteiset menetelmätkin yhä käytössä. [3–5]

Käytössä olevien ohjelmistotuotannon menetelmien runsaudesta voidaan päätellä, ettei ole olemassa mitään sellaista menetelmää, joka sopisi kaikkiin tilanteisiin parhaiten. Yksi ohjelmistonkehityksen ongelmista onkin jokaiselle projektille otollisimman menetelmän valitseminen. Tässä työssä tutkitaan ja pyritään selvittämään kirjallisuudesta muutamien ohjelmistotuotantoprosessien eroja ja niiden valinnan vaikutusta projektiin sekä sitä, minkälaisiin projekteihin nämä prosessit sopisivat parhaiten.

Työssä kuvataan aluksi käytetty tutkimusmenetelmä, minkä jälkeen kolmannessa luvussa käydään läpi vertailtavien ohjelmistotuotantoprosessien esittelyt. Näihin prosesseihin kuuluvat vesiputousmalli, spiraalimalli, iteratiivinen ja inkrementaalinen malli sekä ketterät menetelmät. Neljännessä luvussa ohjelmistotuotantoprosesseja vertaillaan keskenään. Ensin tarkastellaan tuloksia, jotka saatiin prosesseja tai niiden kaltaisia prosesseja käyttämällä eräässä tutkimuksessa, ja sen jälkeen käsitellään prosessien vahvuuksia ja heikkouksia. Viides luku kerää yhteen vertailusta ilmenneet tärkeimmät tulokset ja niistä tehdyt johtopäätökset.

2. TUTKIMUSMENETELMÄ

Tässä työssä tutkimusmenetelmänä toimi lähinnä tiedon hakeminen netistä, koska oman tutkimuksen tekeminen ohjelmistotuotantoprosessien vaikutuksista ja soveltuvuuksista projekteihin ei ollut mahdollista. Tässä luvussa käsitellään tarkemmin käytetyt tiedonhankintamenetelmät.

Tietokantana tiedon hakemissa käytettiin Tampereen teknillisen yliopiston kirjaston Andor-tietokantaa. Kyseinen tietokanta valittiin, koska se sisältää kaikki kirjaston saatavilla olevat tekstit.

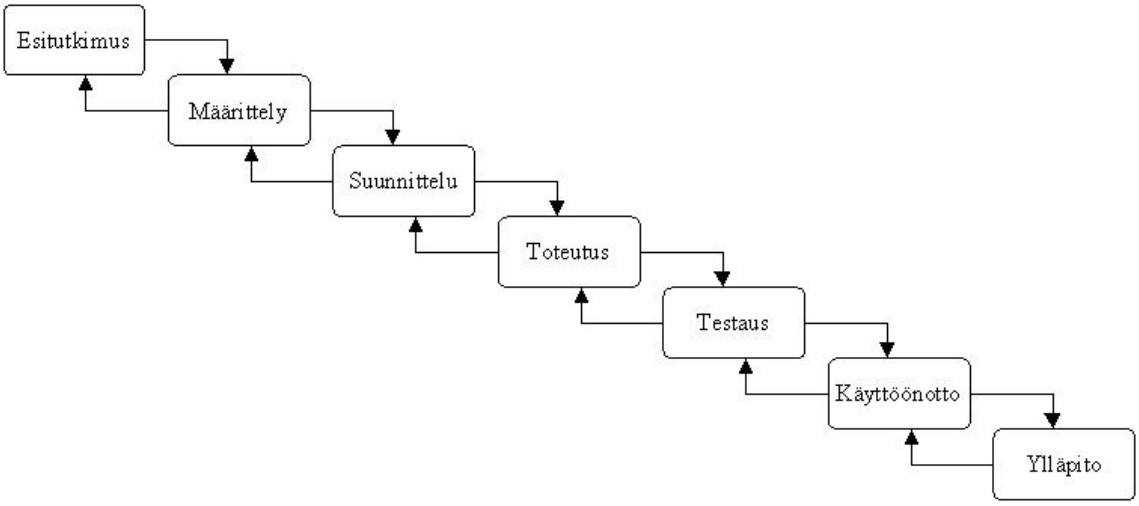
Tiedon etsinnässä tietokannasta hakusanoina tai -lausekkeina käytettiin ohjelmistotuotannon prosesseihin tai menetelmiin viittaavia englanninkielisiä sanoja ja lausekkeita, kuten ”software development methodologies”, comparison AND ”software development methodologies”, ”waterfall model” AND ”agile methods” sekä eri menetelmien nimiä. Hakutuloksia rajattiin ottamalla käyttöön suodattimet ”kokoteksti verkossa” ja ”tieteellinen & vertaisarvioitu”, jotta päästiin käsiksi varmasti luotettaviin ja kokonaisiin teksteihin.

Saaduista vaihtoehdoista valittiin lähinnä otsikoita ja tiivistelmiä lukemalla sopivia tekstejä työtä varten. Koska työssä käsiteltäviä prosesseja empiirisesti vertailevia tutkimuksia oli vaikea löytää, valittiin yksi tällainen tutkimus, joka käytiin läpi tarkemmin. Lisäksi valituista teksteistä oli mahdollista suorittaa myös teoriapohjaisempi prosessien vahvuuksien ja heikkouksien käsittely.

3. PROSESSIEN ESITTELY

3.1 Vesiputousmalli

Vesiputousmalli on vanhin ja tunnetuin elämänkaarimalli ja sitä käytetään laajalti hallituksen projekteissa sekä useissa isoissa yrityksissä. Vesiputousmallille ominaista on sen peräkkäiset vaiheet tai askeleet, joihin lukeutuvat vaatimusten analysointi eli esitutkimus ja määrittely, suunnittelu, toteutus, testaus ja ylläpito. Mallissa käytettävällä runsaalla dokumentoinnilla ja suunnittelulla pyritään estämään suunnitteluvirheiden syntyminen ennen tuotteen kehittämistä. Vesiputousmallin vaiheet eivät ole lomittaisia, eli yksi vaihe alkaa ja loppuu ennen kuin siirrytään seuraavaan vaiheeseen. [1] Mallin vaiheet näkyvät kuvassa 1 havainnollistavasti vesiputouksen rakennetta mukaillen.

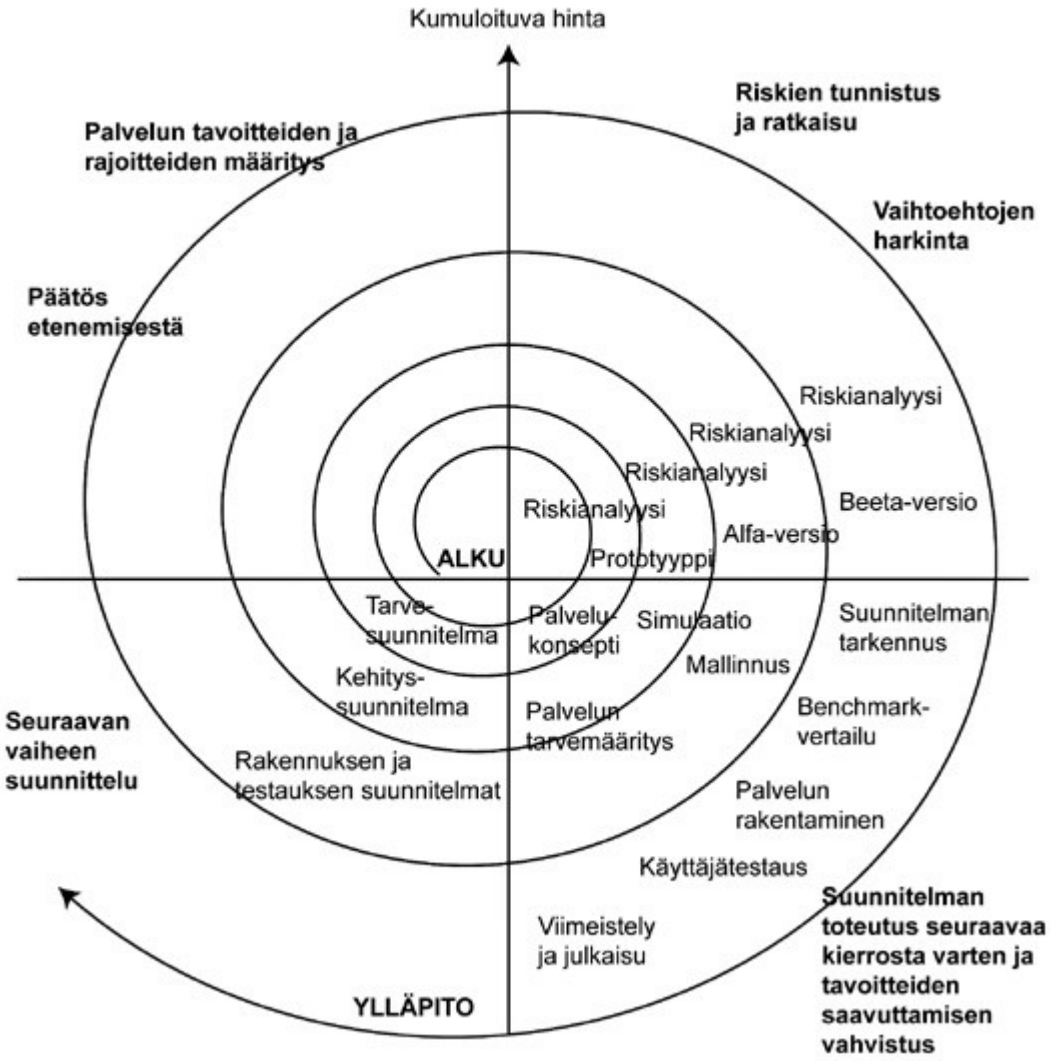


Kuva 1. Vesiputousmallin vaiheet [6].

Vaatimusten analysoinnissa saadaan kuva kehitettävän järjestelmän käyttäytymisestä, toisin sanoen vaatimukset kerätään ja analysoidaan, minkä seurauksena tuotetaan kehitystä tukeva kattava dokumentaatio. Korkean tason suunnittelussa edellisestä vaiheesta kerättyä tietoa käsitellään ja sen pohjalta kaavaillaan toteutettavalle järjestelmälle implementaatio, joka kattaa esimerkiksi käytettävät algoritmit, ohjelmistoarkkitehtuurin ja tietorakenteet. Toteutusvaiheessa kaikki vaatimukset toteutetaan tuotantoympäristöön. Testausvaiheessa testataan koodia ja korjataan siitä löydetyt virheet. Näin varmistetaan, että tuotettu ohjelmistototeutus vastaa alkuperäisiä ohjelmistolle asetettuja vaatimuksia ja että ohjelmisto voidaan ottaa käyttöön. Ylläpitovaihe tulee ohjelmiston julkaisemisen jälkeen ja se pitää sisällään esimerkiksi mahdollisten muutosten, parannusten ja virheiden korjausten tekemisen. [1]

3.2 Spiraalimalli

Spiraalimalli on ohjelmistotuotannon prosessi, jossa yhdistellään vaiheittaista suunnittelua ja prototypointia topdown- ja bottom up -konseptien hyötyjen yhdistämiseksi. Spiraalimalli on niin sanottu metamalli, mikä tarkoittaa, että muut mallit voivat käyttää sitä. Olennaista tälle mallille on riskien arvioimiseen ja projektin riskin pienentämiseen keskittyminen, mikä saavutetaan jakamalla projekti pienempiin osiin. Näin kehitystyön aikana on helpompi tehdä muutoksia ja arvioita riskeistä sekä projektin jatkamisesta koko ohjelmiston elämänkaaren ajan. Spiraalimallissa kehittäjätiimi toteuttaa aluksi pienen osan vaatimuksista kerralla, mikä antaa heille mahdollisuuden oppia uutta tästä alun iteraatiosta riskien analysoinnin kautta. Kehittäjätiimi jatkaa toiminnallisuuden lisäämistä ”spiraaleissa”, kunnes tuote on valmis julkaisu- ja ylläpitovaihetta varten. Spiraalimallin vaiheisiin kuuluvat suunnittelu, riskianalyysi, kehitys ja arviointi. [1] Mallin rakenne ja vaiheet näkyvät havainnollistavassa kuvassa 2.

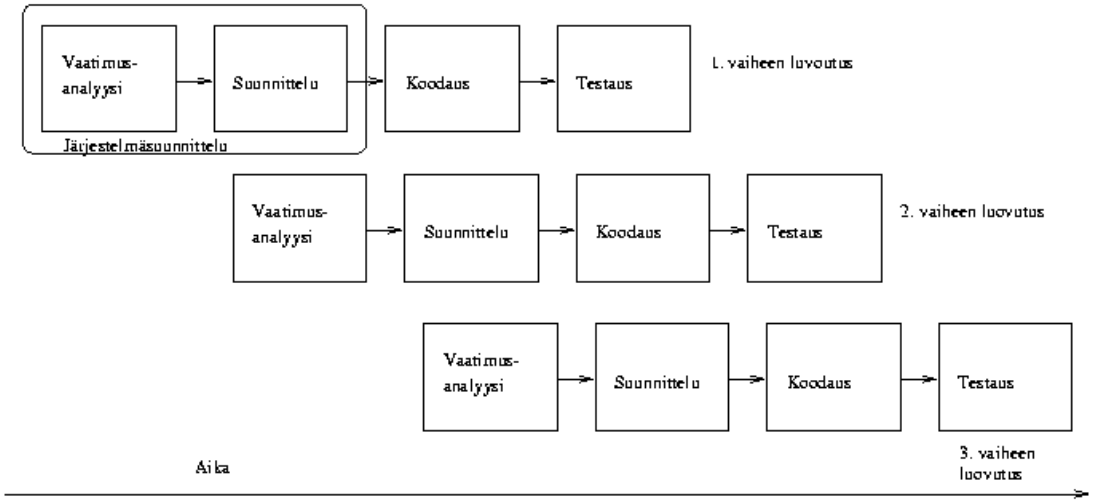


Kuva 2. Spiraalimallin rakenne [7].

Suunnitteluvaiheessa pyritään saamaan ymmärrys kehitettävän järjestelmän vaatimuksista asiakkaiden ja systeemianalyytikkojen välisellä kommunikaatiolla. Riskianalyysivaiheessa tunnistetaan riskejä ja vaihtoehtoisia ratkaisuja ja vaiheen lopuksi tuotetaan prototyyppi. Kehitysvaiheessa ohjelmakoodia kirjoitetaan ja testataan. Arviointivaiheessa asiakas arvioi projektin siihen mennessä aikaan saaman tuotoksen ennen kuin projekti etenee seuraavalle spiraalille. [1]

3.3 Iteratiivinen ja inkrementaalinen malli

Iteratiivinen ja inkrementaalinen malli mukailee perinteistä vesiputousmallia, mutta iteroivalla tavalla. Jokaisen iteraation jälkeen saadaan toimitettava inkrementaatio eli lisäys ohjelmistosta. Ensimmäisessä lisäyksessä toteutetaan perusvaatimukset eli ydintuote. Seuraavissa lisäyksissä lisätään toiminnallisuutta aina edellisen päälle [1]. Kuvassa 3 nähdään havainnollistavasti mallin iteroiva luonne.



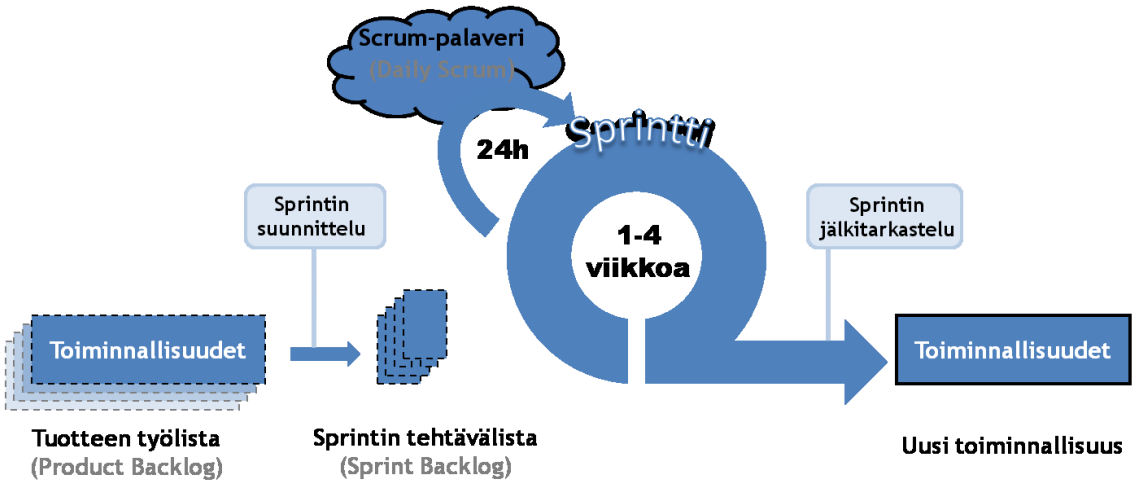
Kuva 3. Iteratiivinen ja inkrementaalinen malli [8].

Jokaiseen iteraatioon kuuluvat vaiheet ovat vaatimusten analysointi, suunnittelu, koodaus, testaus ja iteraation lisäyksen luovutus. Näitä vaiheita toistamalla kaikki suunniteltu toiminnallisuus toteutetaan ja lopputuloksena saadaan valmis tuote [1].

3.4 Ketterät menetelmät

Ketteriä menetelmiä tai ketterää kehittämistä pidetään luovana ja muutoksiin reagoivana työpanoksena asiakkaan tarpeiden vastaamiseksi. Menetelmissä keskitytään siihen, että toimivia ohjelmia saadaan toimitettua nopeammin ja halvemmalla hinnalla. Tyypillisesti

ohjelma kehitetään ja toimitetaan inkrementaaliseen tyyliin, jolloin toimitettavat toiminnallisuudet ovat usein pieniä ja rajoitettuja lyhyisiin toimitusaikoihin. Näin varmistetaan lisättävien toiminnallisuuksien nopean valmistuminen. Ketterälle kehittämiselle on myös olennaista, että käyttäjien osallistumista ohjelman kehitykseen ylläpidetään. [2] Nykyisin yksi varsin suosittu ketterä menetelmä ohjelmistonkehityksessä on Scrum, ja sen määrittelemät vaiheet ketterälle kehittämiselle näkyvät havainnollistavasti kuvassa 4.



Kuva 4. Scrum-menetelmän vaihejako ketterän menetelmän toteuttamiseksi [9].

Kuten tästä kuvasta huomataan, Scrum edustaa hyvin ketterän kehittämisen periaatteita. Uusia toiminnallisuuksia tuotetaan inkrementaalisesti lyhyissä jaksoissa eli sprinteissä, jotka kestävät yhdestä neljään viikkoa. Sprinttien aikana pidetään päivittäisiä Scrum-palavereja, joissa voidaan käydä läpi, mitä on saatu aikaiseksi, ja tehdä suunnitelma seuraavan vuorokauden ajaksi. Sprintin lopuksi pidetään sprintin jälkitarkastelu, jossa kehittäjätiimi ja asiakkaat tai käyttäjät tarkastelevat sprintin aikana tuotettua lisäystä kehitettävään ohjelmistoon. Tuotteen työlista sisältää kaikki ne toiminnallisuudet, jotka kehitettävälle järjestelmälle on vaatimuksissa annettu, ja sprintin tehtävälistaan valitaan aina seuraavan sprintin suunnitteluvaiheessa sopiva määrä näistä toiminnallisuuksista toteuttamista varten. Näin Scrum tukee muutoksiin nopeasti reagoimista ja sopeutumista. Jos esimerkiksi asiakkaan vaatimukset vaihtuvat tai sprintin jälkeisessä tarkastelussa huomataan, että jokin tuotettu toiminnallisuus ei aivan täysin vastaakaan sitä mitä haluttiin, niin voidaan tarvittavat muutokset tehdä nopeasti ja helposti seuraavan sprintin aikana. Asiakkaiden ja kehittäjien välisellä yhteydenpidolla varmistetaankin sitä, ettei järjestelmää kehitetä väärään suuntaan ainakaan kovin suuria osia kerrallaan.

4. PROSESSIEN VERTAILU

Nykyisin on olemassa monenlaisia ohjelmistonkehitykseen tarkoitettuja elämänkaarimalleja ja monet niistä luotiin, jotta päästäisiin eroon perinteiselle vesiputousmallille ominaisista ongelmista. Näitä elämänkaarimalleja eli ohjelmistotuotantoprosesseja on yleensä vaikeaa verrata toisiinsa ja niiden mitattuihin tuloksiin perustuvaa suoraa vertailua on kirjallisuudessa vähän. Tällainen tuotteeseen, projektiin ja ihmisiin kohdistuvien vaikutusten huomioivien mittaustuloksien vertailu mahdollistaa prosessien empiirisen tutkimisen objektiivisesta näkökulmasta. [2]

4.1 Tutkimus

Tarkastellaan tutkimusta, jossa tutkittiin ja mitattiin neljän eri elämänkaarimallin käyttämisen vaikutusta samankaltaisissa projekteissa. Tutkimuksessa mitattiin kattavasti jokaisen tuotettavan ohjelmistotuotteen laatua, kokoa, kehitystehokkuutta ja siihen käytettyä aikaa [2]. Koska projektit olivat samankaltaiset, voitiin tutkimuksessa esiin tulleita tuloksia käyttää hyväksi mallien välisessä vertailussa.

4.1.1 Tutkimuksen esittely

Tutkimuksen tarkoituksena oli tarkastella erilaisten käytettyjen kehityslähestymistapojen vaikutusta lopputuotteeseen ja sen ominaisuuksiin. Käytettyihin neljään ohjelmistokehityksen elämänkaarimalliin kuuluivat V-malli, evoluutionaalinen malli, inkrementaalinen malli ja XP-malli eli extreme programming -malli. [2] V-malli edustaa sekventiaalisia eli jaksollisia ohjelmistotuotantoprosesseja, joihin vesiputousmalli myös kuuluu, ja V-mallia onkin kutsuttu vesiputousmallin kehittyneemmäksi muodoksi. Evoluutionaalinen malli voidaan tämän tutkimuksen yhteydessä rinnastaa aiemmin esiteltyyn spiraalimalliin, joka on yksi evoluutionaalisista malleista. XP-malli puolestaan edustaa ketterän kehityksen menetelmää ja on erittäin samankaltainen aiemmin pintapuolisesti esitellyn Scrumin kanssa.

Tutkimus suoritettiin Islannin yliopistossa vuosien 2003 ja 2004 välisenä talvena osana kokovuotista projektia. Tutkimukseen osallistui 55 opiskelijaa, jotka työskentelivät 15 ryhmässä kehittäen keskenään vertailukelpoisia ohjelmistotuotteita samasta aihe-alueesta. Kaikki tutkimukseen osallistuneet olivat tietotekniikan opiskelijoita viimeisellä vuodellaan yliopistossa, ja osa opiskelijoista oli jo osa-aikaisesti töissä tietotekniikan alalla. [2] Näin ollen voidaan siis olettaa, että kaikilla osallistuvilla opiskelijaryhmillä oli jo tarpeeksi tietoa ja kokemusta ohjelmistojen kehittämisestä, joten tutkimuksen tuloksia voidaan pitää luotettavampina.

Projekteissa kehitettävät tuotteet olivat koti- ja perhekäyttöön tarkoitettuja, tietokantajärjestelmään keskittyviä interaktiivisia paketteja, joita voitiin käyttää nettipohjaisen käyttöliittymän kautta. Tuotteiden käyttäjinä tai asiakkaina toimivat opiskelijat itse sekä heidän perheensä ja ystävänsä. Käyttäjiltä kysyttiin ja saatiin palautetta ohjelmistokehitysprosessin aikana kunkin elämänkaarimallin määräämällä tavalla. Projektien olennaisimmat määritelmät tehtiin etukäteen valmiiksi, jotta voitiin olla varmoja projektien olevan keskenään monimutkaisuudeltaan ja vaikeudeltaan samalla tasolla. [2] Näin projekteissa tapahtuvan ohjelmistonkehityksen seuraaminen ja mittaaminen saatiin tuottamaan mielekästä ja vertailukelpoista dataa eri menetelmien soveltamisen vaikutuksesta. Lisäksi 15 eri projektin ansiosta voitiin olla varmoja, ettei plagioimista tapahtunut ryhmien kesken. Tämä olisi voinut olla mahdollista, jos kaikki 15 ryhmää olisivat kehittäneet saman tuotteen. Ryhmät muodostettiin jakamalla opiskelijat sattumanvaraisesti kolmen tai neljän hengen ryhmiin, ja jokaiselle ryhmälle annettiin projektin aihe sekä käytettävä elämänkaarimalli neljän edellä mainitun mallin joukosta. [2]

Jokaiselle elämänkaarimallille määrättiin asiantuntija, jolta opiskelijat saivat tarvittavat tiedot oman ryhmänsä käyttämästä mallista. Asiantuntija saattoi osallistua oppilaiden tapaamisiin ja antaa pyydettäessä lisäneuvoja mallin mukaisen toiminnan saavuttamiseksi. [2] Mallien oikeanlaisen käytön myötä saatiin näin ollen varmempaa tietoa tutkimuksen tuottamien mittaustulosten oikeellisuudesta. Mitattavaa dataa kerättiin osallistuneilta opiskelijoilta tasaisin väliajoin, jotta kaikki ohjelmiston kehitysprosessin vaiheet saatiin taltioitua. Tämän datan tarkisti ryhmän käyttämästä mallista vastaava asiantuntija, jotta saatiin varmistettua tiedon tarkkuus ja oikeellisuus. [2]

Ryhmien keräämä mittausdata liittyi suoraan käytettyyn elämänkaarimalliin, kehitettävään tuotteeseen ja välituloksiin. Esimerkiksi elämänkaarimalliin liittyen kerättiin tietoa tehdystä työmäärästä, tuotteeseen liittyen kerättiin tietoa koodiriveistä ja välituloksiin liittyen kerättiin tietoa tuotettujen kaavioiden määrästä. Näin tutkimuksessa voitiin keskittyä suurimmaksi osaksi projektiin käytettyihin työmääriin ja tuotteeseen sekä välituloksiin projektin hinnan ollessa vähäpätöisempi huomion kohde, sillä kyseessä oli yliopisto-opiskelijoiden suorittama projekti. [2]

Ennen tutkimuksen tulosten esittelemistä on huomion arvoista mainita, että V-mallia käyttäneet ryhmät kuluttivat mallin määräämällä tavalla paljon ajastaan vaatimusmäärittelyn ja suunnittelun parissa. Näin osa ryhmien suunnittelemista toiminnallisuuksista jäikin ajanpuutteen vuoksi toteuttamatta. Nämä ryhmät aloittivat ohjelmoimisen myöhään ja huomasivat projekteissa käytetyn teknologian aiheuttavan ongelmia, joiden takia ryhmien piti käyttää vielä runsaasti lisää aikaa sopeutuakseen tilanteeseen. Tämä luonnollisesti vaikutti negatiivisesti ryhmien käytössä olevaan aikaan ja kykyyn toimittaa kehitettävä tuote. [2] Vaikka tällä onkin vaikutusta tutkimuksen lopputuloksiin, voidaan tapahtunutta komplikaatiota toisaalta pitää edustavana esimerkkinä sekventiaalisten menetelmien – kuten V-mallin tai vesiputousmallin – joustamattomuudesta odottamattomiin tilannemuutoksiin sopeutumisessa.

4.1.2 Keskimäärin käytetty työmäärä

Ryhmien projektiin käyttämä aika mitattiin tunneissa ja se jaettiin seitsemään yleiseen aktiviteettiin: vaatimusmäärittelyyn, suunnitteluun, ohjelmointiin, testaukseen, tarkasteluun, korjaukseen eli koodin korjaukseen ja refaktorointiin sekä muihin asioihin, kuten teknologiaan tutustumiseen [2]. Ryhmien keskimääräinen työpanos tunteina näissä seitsemässä osa-alueessa nähdään taulukosta 1, jossa osa-alueet ovat sarakkeissa vastaavassa järjestyksessä vasemmalta oikealle. Lisäksi taulukkoon on lisätty loppuun sarakkeet projektiin käytetyille kokonaistunneille ja projektikuukausille (lyhenne PM tulee sanoista project month eli projektikuukausi), joka saadaan jakamalla kokonaistunnit 152 tunnilla. Ryhmien elämänkaarimallit on esitetty riveittäin ja niitä kuvaavista lyhenteistä VM tarkoittaa V-mallia, EM evoluutionaalista mallia, IM inkrementaalista mallia ja XP extreme programmingia. Viimeisellä rivillä (OAve eli overall average) on esitetty eri elämänkaarimalleissa käytettyjen tuntimäärien keskiarvot osa-alueittain.

Taulukko 1. Ryhmien keskimääräinen työpanos tunteina [2].

Group	ReqsSpec	Design	Code	Integration and testing	Review	Repair	Other	Total hours	Total PM
VM	73.9	68.6	206.1	35.6	56.9	60.4	246.5	748.0	4.92
EM	67.8	58.0	169.1	57.8	23.0	48.0	125.6	549.3	3.61
IM	43.8	51.2	185.7	40.7	37.7	53.8	121.6	534.5	3.52
XP	16.2	26.2	205.6	82.7	46.9	92.7	122.4	592.5	3.90
OAve	53.3	52.8	191.1	53.1	41.0	62.4	158.6	612.1	4.03

Kuten taulukosta nähdään, V-mallissa käytetty aika vaatimusmäärittelyyn ja suunnitteluun oli muita suurempi. Varsinkin ero XP:n kanssa oli erityisen suuri: V-mallin keskimääräinen aika vaatimusmäärittelylle oli 73,9 tuntia ja suunnittelulle 68,6 tuntia, kun taas XP:n ajat vastaaville osa-alueille olivat ainoastaan 16,2 ja 26,2 tuntia. Tämä havainnollistaa hyvin sekventiaalisten menetelmien keskittymistä perinpohjaiseen asiakasvaatimusten selvittämiseen ja huolellisen suunnittelun tekemiseen. Ohjelmoinnin suhteen elämänkaarimalleilla ei ollut kovin suuria keskinäisiä eroja tehdyssä työmäärässä, mutta integraatiota ja testausta tarkastellessa huomataan XP:tä käyttäneiden ryhmien käyttäneen keskiarvoisesti eniten aikaa tässä osa-alueessa. Syynä tähän saattaa olla vähäinen panostus vaatimusten tutkimiseen ja suunnitteluun, ja suurempi panostus muissa malleissa näihin osa-alueisiin saattoi auttaa virheiden aikaisemmassa havaitsemisessa. Korjauksiin ku-

lunut suurempi aika XP:ssä muihin malleihin verrattuna johtunee samasta syystä. Taulukosta nähdään myös, että V-mallia käyttäneillä ryhmillä kului keskimäärin puolet enemmän aikaa muissa asioissa, jotka eivät suoranaisesti liity elämänkaarimalliin. Tämä kuvastaakin aiemmin mainittuja V-malliryhmien ongelmia käytetyn teknologian kanssa ja tarvetta sopeuta niihin.

Yhden opiskelijan tekemän työn määrä ryhmissä vaihteli runsaasti, ja joissakin ryhmissä opiskelijoita oli vain kolme, kun taas useimmissa ryhmissä oli neljä opiskelijaa. Näiden seikkojen takia onkin mielekkäämpää tutkia työmäärän prosentuaalista jakautumista eri elämänkaarimallien vertailua ajatellen. [2] Tämän prosentuaalinen työmäärän jakautuminen näkyykin taulukossa 2 samaan tyyliin esitettynä kuin työmäärä tunneittain aiemmassa taulukossa 1.

Taulukko 2. Ryhmien keskimääräinen työpanos prosentteina [2].

Group	ReqsSpec, %	Design, %	Code, %	Integration and testing, %	Review, %	Repair, %	Other, %	Total, %
VM	10	10	27	5	7	8	33	100
EM	12	10	29	11	5	9	23	99
IM	8	9	35	7	7	10	23	99
XP	3	4	34	14	8	17	21	101
OAve	8.9	8.9	30.8	8.9	6.5	10.0	25.7	100

Taulukosta voidaan havaita sama kuin edellisestäkin taulukosta eli se, että XP:n keskimääräinen työmäärä vaatimusmäärittelylle ja suunnittelulle oli pieni verrattuna muihin malleihin. Erityisesti eron huomaa V-malliin ja evoluutionaaliseen malliin, jolla näyttäisi olleen keskimääräisesti suurin suhteellinen työpanos vaatimusmäärittelyssä. Muilta osin taulukon tulosten perusteella näyttäisi siltä, että eri mallien suhteellisissa työmäärissä ei ollut kovin paljon eroa.

4.1.3 Vaatimukset ja korkean tason suunnittelu

Ryhmissä toteutettua vaatimusmäärittelyä ja korkean tason suunnittelua mitattiin vaatimusmäärittelyn sivujen määrällä, käyttötapauksilla, käyttäjäinteraktiokuvilla tietokantakaavioilla ja tietokantatauluilla [2]. Näiden tuotettujen dokumenttien keskiarvoinen määrä elinkaarimalleittain näkyy taulukossa 3, jossa mitatut osa-alueet näkyvät vastaavassa järjestyksessä vasemmalta oikealle.

Taulukko 3. Ryhmien keskimääräiset vaatimusmäärittelyn ja korkean tason suunnittelun tulokset [2].

Group	Pages	Use-cases	Screens	Database diagrams	Database tables
VM	22.0	7.5	6.8	1.0	3.3
EM	19.0	14.0	11.0	1.3	6.8
IM	22.7	14.7	5.0	1.3	4.3
XP	5.7	8.3	17.7	0.7	5.0
OAve	17.8	11.1	9.9	1.1	4.9

Kuten taulukosta nähdään, XP:tä käyttävät ryhmät tuottivat muita elämänkaarimalleja käyttäviä ryhmiä huomattavasti vähemmän vaatimusmäärittelyn sivuja. Tämä tulos edustaa hyvin ketteriä menetelmiä, joissa alkudokumentaation tekoon panostetaan muita menetelmiä vähemmän. Taulukosta huomataan myös, että evoluutionaalisen ja inkrementaalisen mallin ryhmät tuottivat keskimäärin muita enemmän käyttötapauksia, ja että XP:n käyttäjäinteraktiokuvien määrä oli varsinkin V-mallin ja inkrementaalisen mallin vastaavia määriä suurempi.

4.1.4 Suunnittelu

Ryhmissä tehtyä suunnittelua mitattiin suunnittelukaavioiden, kuten yleiskatsaus-, luokka-, sekvenssi-, tilakaavioiden ja muiden kaavioiden määrällä [2]. Näiden suunnittelukaavioiden keskiarvoiset määrät elämänkaarimalleittain näkyvät taulukossa 4, jossa edellä luetellut suunnittelukaaviot ovat esitettyinä sarakkeissa vastaavassa järjestyksessä vasemmalta oikealle.

Taulukko 4. Ryhmien keskiarvoiset suunnittelukaavioiden määrät [2].

Group	Overview diagrams	Class diagrams	Sequence diagrams	State diagrams	Other diagrams	Total
VM	1.0	2.5	2.3	5.3	1.8	12.8
EM	1.3	1.5	3.8	0.5	1.5	8.5
IM	2.3	3.7	7.7	0.0	2.0	15.7
XP	1.0	0.0	0.0	0.0	0.7	1.7
OAve	1.4	1.9	3.4	1.6	1.5	9.8

Kuten taulukosta nähdään, eri kaavioiden määrät olivat melko pienet jokaisella elämänkaarimallilla. Kuitenkin kaavioiden kokonaismäärää tarkastelemalla huomataan, että XP:tä käyttäneet ryhmät tuottivat kaavioita keskimäärin paljon muita malleja käyttäneitä ryhmiä vähemmän, koskien erityisesti inkrementaalista mallia ja V-mallia.

4.1.5 Ohjelmiston koko

Ryhmien kehittämien ohjelmistojen kokoja mitattiin projekteissa käytetyn Java-kielen luokkien ja koodirivien määrillä. Koodirivien kokonaismäärä sisälsi Javalla tuotettujen koodirivien lisäksi myös JSP- ja XML-kielillä sekä muilla skriptauskielillä tuotetut koodirivit [2]. Nämä mitatut keskimääräiset tulokset näkyvät elämänkaarimalleittain taulukossa 5, jossa edellä luetellut kielet ovat vastaavassa järjestyksessä vasemmalta oikealle.

Taulukko 5. Ryhmien keskimäärin tuottamat luokat ja koodirivit. [2]

Group	Java classes	Java	JSP	XML	Other	Total LOC
VM	8.5	1032	1161	13	27	2233
EM	26.8	1477	1903	0	37	3417
IM	20.7	1803	922	36	14	2776
XP	27.7	4836	1662	987	254	7740
OAve	20.4	2140	1429	223	76	3867

Kuten taulukosta nähdään, XP:tä käyttäneet ryhmät tuottivat keskimäärin selvästi enemmän koodirivejä kuin missään muussa elämänkaarimallissa. Koodirivien määrässä suurin ero huomattiin Java-kielen koodiriveissä, kun taas muiden kielien koodirivien määrät olivat melko samankokoiset mallien välillä. Huomioitakoon, että V-malliryhmät tuottivat huomattavasti vähemmän luokkia kuin muiden mallien ryhmät. Tuloksista voidaan päätellä, että ohjelman kokoa ajatellen tuottavuudeltaan paras lähestymistapa tuotteen kehittämiseen oli XP ja tuottavuudeltaan huonoin V-malli.

4.1.6 Tutkimuksen tulokset tiivistettynä

Tutkimuksen tulosten perusteella voidaan todeta, että elämänkaarimalleista tehokkain koodirivien tuottamiseen oli ketteriä menetelmiä edustava XP. Dokumentaatiota ja kaavioita XP ei kuitenkaan tuottanut yhtä paljon kuin muut menetelmät, mikä näkyykin sen vähäisessä työpanoksessa dokumentaation tekemiseen. Evoluutionaalinen malli ja inkrementaalinen malli olivat monin tavoin samankaltaisia V-mallin kanssa, mutta myös joitakin eroja löytyi, esimerkiksi käyttötapauksia tuotettiin evoluutionaalisessa ja inkrementaalisessa mallissa paljon enemmän kuin V-mallissa ja XP:ssä. V-malli näyttäisi myös pärjäävän hieman huonommin evoluutionaaliselle ja inkrementaaliselle mallille koodin tuottavuudessa.

Vaikka tämän yhdentyyppisiin projekteihin keskittyneen tutkimuksen perusteella ei voidakaan julistaa ehdottomasti parasta lähestymistapaa ohjelmistojen kehittämiseen, antavat tutkimuksen tulokset kuitenkin jonkinlaista suuntaa menetelmien keskinäiselle paremmuudelle tietyissä tilanteissa. Tulosten perusteella näyttäisi siltä, että menetelmäksi kannattaisi valita XP tai jokin muu ketterän kehityksen menetelmä, jos nopea toiminnallisuuden lisääminen on tärkeää eikä dokumentaatiota tarvita runsaasti. Jos taas asiakasvaatimukset ovat selvillä, huolellinen suunnittelu ja dokumentointi ovat tärkeässä asemassa ja aikaa on tarpeeksi, voisi parempi vaihtoehto menetelmäksi olla jokin kolmesta muusta menetelmätyypistä. Näistä kolmesta evoluutionaalinen ja inkrementaalinen malli vaikuttavat kuitenkin olevan hieman sietokykyisempiä odottamattomien muutosten suhteen, minkä takia ne sijoittuvatkin jonnekin sekventiaalisten ja ketterien menetelmien välille.

4.2 Prosessien vahvuudet ja heikkoudet

Tarkastellaan esiteltyjen ohjelmistotuotantoprosessien vahvuuksia ja heikkouksia. Esitetään myös vaihtoehtoja sopiville tilanteille käyttää kutakin prosessia.

4.2.1 Vesiputousmalli

Vesiputousmallin vahvuuksia ovat sen helppo ymmärtäminen ja toteuttaminen. Malli on vanha ja erittäin tunnettu, mikä vähentää tarvetta tutustua sen toimintamalliin. Helppo

toteuttaminen johtuu lähinnä siitä, että malli on lineaarinen eli vaiheet prosessoidaan ja suoritetaan yksi kerrallaan. Vesiputousmalli toimii hyvin kypsissä tuotteissa ja se tarjoaa selvää rakennetta kokemattomille kehittäjätiimeille. Lisäksi ohjelmointiosuutta edeltävät mittavat vaatimusanalyysi- ja suunnitteluvaiheet vähentävät suunnittelukustannuksia ja auttavat virheiden ehkäisyssä. [1]

Vesiputousmallin heikkouksiin kuuluvat joustamattomuus ja se, että kaikki vaatimukset on tiedettävä etukäteen. Joustamattomuuden takia on virheen korjaamiseksi hankalaa palata takaisin testausvaiheesta. Tästä seuraakin riskejä ja epävarmuutta, koska pienikin virhe tai muutos kehitettävässä ohjelmistossa voi aiheuttaa paljon lisävaivaa projektissa. Tilanne voi vielä huonontua, mikäli mallia käytetään asiakkaan ollessa epävarma haluamistaan ominaisuuksista. Lisäksi mallissa asiakkailla voi olla vain vähän mahdollisuuksia tarkastella kehitettävää tuotetta ennen kuin se on edennyt jo pitkälle tuotantoprosessissa. [1]

Vesiputousmallin käyttäminen voi olla kannattavaa, kun laatu on hintaa ja aikataulua tärkeämpää tai jos asiakasvaatimukset ovat kaikki tunnettuja, selkeitä ja pysyviä. Myös valmiin tuotteen uuden version tuottamisessa tai valmiin tuotteen uudelle alustalle siirtämisessä saattaa olla hyvä harkita vesiputousmallin käyttämistä. [1]

4.2.2 Spiraalimalli

Spiraalimallin hyviin puoliin kuuluu riskianalyysin paljous ja ohjelman tuottaminen sen elämänkaaren aikaisessa vaiheessa. Muita hyviä ominaisuuksia ovat esimerkiksi hyväksymisen ja dokumentaation tarkka kontrolli, mahdollisuus lisätä toiminnallisuutta myöhemmin, projektin tarkkailun helppous sekä käyttäjiltä tiheään saatava aikainen palaute. Malli ilmaisee ajoissa ylitsepääsemättömistä riskeistä, ja prototypointityökalujen avulla voidaan tarkastella etukäteen jokaista vaihetta erikseen. [1]

Spiraalimallin heikkouksiin kuuluvat yleensä mallin kallis hinta ja riskien arvioimiseen tarvittava ammattitaito. Pienissä tai vähäriskisissä projekteissa riskianalyysiin kuluva aika voi kasvaa liian suureksi, ja ajankäyttö voi muutenkin paisua mallin aktiviteeteista, kuten suunnittelusta, riskianalyysistä ja prototypoinnista. Projektin onnistuminen saattaa myös riippua liikaa riskianalyysin onnistumisesta. Lisäksi projektin hallinnasta voi tulla monimutkaista välivaiheiden vaatiman dokumentaatiomäärän takia. [1]

Spiraalimallia on sopiva käyttää projekteissa, joissa riskit ovat keskitasoa tai korkeita ja riskien arviointi on tärkeää. Malli sopii myös tilanteisiin, joissa asiakas ei ole varma haluamistaan ominaisuuksista, ja joissa suuria muutoksia ohjelmaan voidaan odottaa. [1]

4.2.3 Iteratiivinen ja inkrementaalinen malli

Iteratiivisen ja inkrementaalisen mallin vahvuuksiin kuuluu eritoten toiminnallisuuksien kehittäminen pienemmissä lisäyksissä. Näin kaikkein tärkeimmät toiminnallisuudet voidaan toteuttaa ensimmäiseksi, ja epäonnistumisen sekä muuttuvien vaatimusten aiheuttama riski jakautuu pienemmille osasille koko projektin sijasta. Lisäksi toimitettavien lisäysten toimittaminen alussa on nopeaa, ja asiakkaat saavat ajoissa tärkeimmät toiminnallisuudet, jolloin he pääsevät vaikuttamaan seuraavaan iteraatioon. [1]

Huonona puolena malli vaatii hyvän suunnittelun ja aikaisen määrittelyn valmiista ja toimivasta järjestelmästä, jotta toteutettavat lisäykset voidaan määritellä. Lisäksi mallissa ei ole mahdollista suorittaa iteraatioita lisäyksen sisällä. [1]

Malli sopii projekteihin, joissa riskit ovat pieniä tai keskitasoisia. Se sopii myös projekteihin, joissa on tarve saada perustoiminnallisuutta markkinoille nopeasti sekä projekteihin, joilla on pitkä kehitysaikataulu. Malli käy myös tilanteisiin, joissa käyttäjän annetaan totutella kehitettävän järjestelmän uuteen teknologiaan pienissä askelissa. Lisäksi mallia voidaan käyttää, jos kokonaisen systeemin kerralla valmistaminen on liian riskialtista. [1]

4.2.4 Ketterät menetelmät

Ketterien menetelmien vahvuuksiin kuuluu keskittyminen ohjelmien nopeaan ja halpaan toimittamiseen. Uusia toiminnallisuuksia tuotetaan pienissä osissa ja niiden kehittäminen tapahtuu lyhyissä jaksoissa, jolloin toiminnallisuudet saadaan nopeasti toimitettua. Ketterissä menetelmissä korostetaan jatkuvan palautteen saamista asiakkailta, jotka ovat tiheästi yhteydessä kehittäjien kanssa. Näin voidaan sopeutua muuttuviin vaatimuksiin nopeasti. [2]

Ketterien menetelmien heikkouksiin saattaa joissakin tilanteissa kuulua dokumentaation vähyys. Tällöin esimerkiksi ohjelmiston ylläpito tai koodin toiminnallisuuden tarkastelu voi olla hankalampaa. Myös ohjelman rakenne voi olla huono, mikäli alussa ei panosteta tarpeeksi suunnitteluun.

Ketterät menetelmät sopivat hyvin projekteihin, joissa vaaditaan asiakkaiden ja kehittäjien välistä tiivistä kommunikaatiota. Tällaisissa projekteissa voidaan olettaa vaatimusten olevan vielä epäselviä ja mahdollisesti vaihtuvia, ja ketterien menetelmien vahvuutena onkin juuri nopea muutoksiin reagointi. Lisäksi ketterien menetelmien käyttö on hyvä vaihtoehto, kun tuote pitää saada valmiiksi nopeasti ja edullisesti.

5. YHTEENVETO

Tässä työssä käsiteltiin ohjelmistotuotantoprosessien eroja ja pyrittiin selvittämään niiden valinnasta aiheutuvaa vaikutusta projektiin sekä sitä, minkälaisiin projekteihin nämä prosessit sopisivat parhaiten. Käsiteltäviä ohjelmistotuotantoprosesseja valittiin neljä ja niihin kuuluivat vesiputousmalli, spiraalimalli, iteratiiviset ja inkrementaaliset menetelmät sekä ketterät menetelmät.

Aluksi työssä esiteltiin käytettävä tutkimusmenetelmä, jonka jälkeen siirryttiin käsiteltävien ohjelmistotuotantoprosessien esittelyyn. Tämän jälkeen siirryttiin tarkastelemaan tutkimusta, jossa valitut prosessit tai vähintään niitä vastaavat prosessit olivat olleet käytössä. Tutkimus esiteltiin, ja sen jälkeen esiteltiin tutkimuksessa ilmenneet mittaustulokset. Tuloksia analysoitaessa havaittiin, että ketteriä menetelmiä käyttämällä dokumentaatiota ei kirjoiteta yhtä paljon kuin muita prosesseja käyttämällä, ja että ketterillä menetelmillä saadaan muita prosesseja nopeammin tuotettua enemmän ohjelmakoodia. Tutkimuksen jälkeen siirryttiin käsittelemään teoreettisemmalta pohjalta prosessien vahvuuksia ja heikkouksia sekä niiden sopivuutta erilaisiin projekteihin. Tästä saatiin samankaltaisia tuloksia kuin tutkimuksestakin. Lisäksi havaittiin, että vesiputousmalli sopii laatukriittisempiin projekteihin, joissa vaatimukset ovat selvillä ja muuttumattomia, spiraalimalli sopii projekteihin, joissa riskien arvioiminen on tärkeää ja vaatimukset voivat muuttua, iteratiivinen ja inkrementaalinen malli sopii pitkäkestoisiin projekteihin ja projekteihin, joissa on tärkeä saada perustoiminnallisuutta nopeasti markkinoille ja ketterät menetelmät sopivat projekteihin, joissa vaatimukset ovat epäselviä ja voivat muuttua sekä asiakkaiden ja käyttäjien välinen kommunikaatio on tärkeässä asemassa.

Tutkimuksen tuloksista sekä prosessien vahvuuksien ja heikkouksien käsittelystä huolimatta ei voida täysin varmasti sanomaan, että tietyn prosessin valitsemalla vaikutukset projektissa olisivat tämän työn havaintojen mukaiset. Oikean prosessin valinta ja sen vaikutukset riippuvat aina erittäin voimakkaasti toteutettavasta projektista ja erilaisista osatekijöistä, kuten rahasta tai vaikkapa työntekijöiden määrästä. Tämän työn tulokset antavatkin lähinnä suuntaa oikeanlaisen prosessin valitsemiseen ja sen projektiin kohdistamien vaikutusten arvioimiseen.

LÄHDELUETTELO

[1] A. Alshamrani, A. Bahattab, A Comparison Between Three SDLC Models Waterfall Model, Spiral Model, and Incremental/Iterative Model, International Journal of Computer Science Issues (IJCSI), Vol. 12, Iss. 1, 2015, pp. 106.

[2] O. Benediktsson, D. Dalcher, H. Thorbergsson, Comparison of software development life cycles: a multiproject experiment, IEE Proceedings - Software, Vol. 153, Iss. 3, 2006, pp. 87.

[3] R.H. Kulkarni, P. Padmanabham, M. Harshe, K.K. Baseer, P. Patil, Investigating Agile Adaptation for Project Development, International Journal of Electrical and Computer Engineering, Vol. 7, Iss. 3, 2017, pp. 1278.

[4] K.T. Ryan, Software processes for a changing world, Journal of Software: Evolution and Process, Vol. 28, Iss. 4, 2016, pp. 236–240.

[5] L.R. Vijayasarathy, C.W. Butler, Choice of Software Development Methodologies: Do Organizational, Project, and Team Characteristics Matter? IEEE Software, Vol. 33, Iss. 5, 2016, pp. 86–94.

[6] Vesiputousmallin kuva, saatavissa: http://www.okol.org/verkkokurssit/datanomi/tietojarjestelmien_kaytto_ja_kehittaminen/johdatus_tietojarjestelmiin/images/kuva2_2.jpg (viitattu 23.10.2017)

[7] Spiraalimallin kuva, saatavissa: http://www.uiah.fi/ISBN/951-558-172-9/pics/image031.jpg (viitattu 23.10.2017)

[8] Iteratiivisen ja inkrementaalisen mallin kuva, saatavissa: https://www.cs.helsinki.fi/u/taina/ohjelmistotuotanto/luennot/k99/prosessi/lisaava.gif (viitattu 6.11.2017)

[9] Ketterän menetelmän (Scrum) kuva, saatavissa: https://hlab.ee.tut.fi/hmopetus/system/files/u372/scrum.png (viitattu 26.11.2017)